Decentralized Recommendation based on Matrix Factorization: A Comparison of Gossip and Federated Learning*

István Hegedűs[1][0000−0002−5356−2192], Gábor Danner[1][0000−0002−9983−1060], and Márk Jelasity[1,2][0000−0001−9363−1482]

[1] University of Szeged, Szeged, Hungary
[2] MTA SZTE Research Group on Artificial Intelligence, Szeged, Hungary

Abstract. Federated learning is a well-known machine learning approach over edge devices with relatively limited resources, such as mobile phones. A key feature of the approach is that no data is collected centrally; instead, data remains private and only models are communicated between a server and the devices. Gossip learning has a similar application domain; it also assumes that all the data remains private, but it requires no aggregation server or any central component. However—one would assume—gossip learning must pay a price for the extra robustness and lower maintenance cost it provides due to its fully decentralized design. Here, we examine this natural assumption empirically. The application we focus on is making recommendations based on private logs of user activity, such as viewing or browsing history. We apply low rank matrix decomposition to implement a common collaborative filtering method. First, we present similar algorithms for both frameworks to efficiently solve this problem without revealing any raw data or any user-specific parts of the model. We then examine the aggregated cost in both cases for several algorithm-variants in various simulation scenarios. These scenarios include a real churn trace collected over mobile phones. Perhaps surprisingly, gossip learning is comparable to federated learning in all the scenarios and, especially in large networks, it can even outperform federated learning when the same subsampling-based compression technique is applied in both frameworks.

1 Introduction

Mobile phones represent a key source of data and a very important platform not only for running pre-trained models but also for learning [17]. This is because collecting data centrally has become more and more problematic over the past

* In Springer CCIS 1167 (Proc. DMLE 2019). The final authenticated version is available online at https://doi.org/10.1007/978-3-030-43823-4_27. This study was supported by the Hungarian Government and the European Regional Development Fund under the grant number GINOP-2.3.2-15-2016-00037 ("Internet of Living Things") and by the Hungarian Ministry of Human Capacities (grant 20391-3/2018/FEKUSTRAT).

few years due to novel data protection rules [7] as well as the increasing public awareness to privacy issues. For this reason, there is an increasing interest in methods that keep the raw data on the device and process it using distributed algorithms.

Google introduced *federated learning* to answer this challenge [10, 12]. Not unlike the well-known parameter server architecture [6], a server maintains the current model and regularly distributes it to the workers who in turn calculate a gradient update and send it back to the server, where the updates are aggregated. In federated learning, this framework is optimized so as to minimize communication between the server and the workers. For this reason, the local update calculation is more thorough, and compression techniques can be applied when uploading the updates to the server. *Gossip learning* has also been proposed to address the same challenge [9, 14]. This approach is fully decentralized, no parameter server is necessary. Nodes exchange and aggregate models directly. Since no infrastructure is required, and there is no single point of failure, gossip learning enjoys a significantly *cheaper scalability and better robustness* than centralized approaches.

However, it is not clear whether gossip learning is competitive in terms of convergence time and communication cost. To shed light on this question, we carry out an empirical comparison of the two approaches. To do this, we implement a recommender system in both paradigms, based on low-rank matrix decomposition. The gossip learning implementation is based on our previous work [9]. We propose a federated learning implementation as well, following the same design, but adapted to the centralized communication pattern. Also, inspired by [10], we apply subsampling to reduce communication in both approaches.

The result of our comparison is that gossip learning is in general comparable to the centrally coordinated federated learning approach, and in some scenarios it actually outperforms federated learning. One should obviously not jump to conclusions based on one empirical study, but our results suggest that fully decentralized algorithms perhaps deserve more attention in the future.

To sum up our key original contributions in the present study: (1) we propose an efficient collaborative filtering method for federated learning; (2) we improve several details of our previous solution [9] as well including the introduction of coordinate-based age parameters to manage aggregation and the application of an optimized version of subsampling to gossip learning; and (3) we compare the two methods empirically based on a realistic churn trace collected by the application Stunner [2].

We are aware of only two (at the time of writing, unpublished) studies that address the specific problem of recommender systems in federated learning. The first is based on the idea of meta-learning [4]. Here, it is assumed that the devices have enough data to learn a model based only on local data. Then, federated learning is used to find the optimal hyperparameters for the algorithm, using the devices to calculate gradients for the hyperparameters. We are interested in scenarios where there is not much local data, so meta-learning is not an option. The second study is closer to our approach [1] in spirit. However the authors as-

sume a different setup with only implicit binary feedback as data (e.g., a movie was watched or not). Due to this, their input data is a dense matrix (there are no ratings labeled as "unknown") so compressed communication is more problematic. We focus on modeling only the known ratings (a small minority of all ratings) and make predictions based on these. Also, the optimization algorithm they approximate is alternating least squares with a federated gradient optimization step in the inner loop, while we use simple SGD which is more robust to failure and asynchrony.

We note that both approaches offer mechanisms for explicit privacy protection, apart from the basic feature of not collecting data. In federated learning, Bonawitz et al. [3] describe a secure aggregation protocol, whereas for gossip learning one can apply the methods described in [5]. Here, we are concerned only with the efficiency of the different communication patterns and do not compare security mechanisms.

The outline of the paper is as follows. In Section 2, we describe the low rank matrix decomposition problem, formulated as a machine learning problem. Here, we also present the key ideas to solve this problem in a decentralized setting. In Section 3, we describe the basics of solving the problem with federated learning while in Section 4 we present the gossip learning algorithm for the same problem. In Section 5 we present the key details of the learning algorithm that are common to both approaches. These include the details of the update rule, the subsampling technique, and the initialization. Finally, in Section 6, we present our empirical results.

2 Rank-k matrix approximation

Here, we present the problem definition in the form of a model and a corresponding loss function. We also describe our approach and main assumptions—common to both federated learning and gossip learning—regarding the optimization of the model.

2.1 Problem definition

The problem of rank-k matrix approximation [11] is defined in the following way. Let $A \in \mathbb{R}^{m \times n}$ be a matrix that contains our data (for example, user ratings of items such as movies, songs, or locations). The goal is to find two matrices $X \in \mathbb{R}^{m \times k}$ and $Y \in \mathbb{R}^{n \times k}$ that minimize the error function

$$J(X, Y) = \frac{1}{2} \left\| A - XY^T \right\|_F^2 = \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{n} (a_{ij} - \sum_{l=1}^{k} x_{il} y_{jl})^2. \tag{1}$$

We consider the matrix XY^T an optimal rank-k approximation of A. Note that the rank of X and Y^T (and therefore XY^T) is at most k. Usually a k much smaller than m and n is chosen to significantly compress the data. X and Y^T

can be interpreted as high level features (e.g. genres of movies and tastes of users) that compactly represent the original data.

Often in practice we have only partial information regarding A; that is, some values in A might be unknown. As an important generalization of the problem above, here we are looking for a rank-k decomposition that approximates only the known values. A common approach is to minimize the error function

$$J(X,Y) = \frac{1}{2} \sum_{(i,j)\in I} (a_{ij} - \sum_{l=1}^{k} x_{il} y_{jl})^2 + \frac{\lambda}{2} \|X\|_F^2 + \frac{\lambda}{2} \|Y\|_F^2 , \tag{2}$$

where I contains the indices of the known values of A. We can then use the decomposition XY^T to approximate the unknown values in A, since XY^T is fully defined. Note the here we also included additional regularization terms and the regularization parameter λ. This helps stabilize the optimization process in a machine learning context.

Another practical technique used is to add bias terms to the model. The bias terms $b \in \mathbb{R}^{m\times 1}$ and $c \in \mathbb{R}^{1\times n}$ are incorporated into the model via the loss function

$$J(X,Y,b,c) = \frac{1}{2} \sum_{(i,j)\in I} (a_{ij} - b_i - c_j - \sum_{l=1}^{k} x_{il} y_{jl})^2 + \frac{\lambda}{2} \|X\|_F^2 + \frac{\lambda}{2} \|Y\|_F^2 . \tag{3}$$

For example, in a recommender system, the bias can represent the fact that some users tend to give higher or lower scores than others, and some movies tend to get higher or lower scores. Intuitively, the bias represents average scores, and X and Y represent relative differences. This often enhances the prediction performance. With bias, the approximation of A (both known and unknown values) is given by $XY^T + b\mathbf{1}_n + c^T\mathbf{1}_m$ where $\mathbf{1}_k$ is a row vector of k ones.

2.2 Optimization approach

Our targeted application environment consists of a potentially large set of personal devices holding private data. We follow the approach in our previous paper [9] and we will adapt the same approach to federated learning. We shall assume that each row in matrix A is stored on exactly one device. We shall also assume that each device will host exactly one row. This setup covers applications where one row of the matrix belongs to one user and the devices belong to exactly one user, as in the case of mobile phones. One matrix row can naturally represent any kind of private user activity, such as watching movies. We should add though that if more than one row is stored on a device, the algorithms are still applicable.

The main idea is that matrix X will also be stored in a similar manner; that is, every device will store the row of X that belongs to the row of A stored on the device. This way, the matrix X that contains information about the users is completely private, every device knows only its own row. However, the entire

Algorithm 1 Federated Learning Master

1: $(t, Y, c) \leftarrow \text{initY}()$
2: **loop**
3: $(\widetilde{t}, \widetilde{Y}, \widetilde{c}) \leftarrow (0_n, 0_{n,k}, 0_n)$
4: **for** every node i **in parallel do** ▷ non-blocking (in separate thread(s))
5: send (t, Y, c) to i
6: receive (t', Y', c') from i ▷ model gradient
7: $(\widetilde{t}, \widetilde{Y}, \widetilde{c}) \leftarrow (\widetilde{t} + t', \widetilde{Y} + Y', \widetilde{c} + c')$
8: **end for**
9: wait(Δ_f) ▷ the round length
10: **for** $j \leftarrow 1 \dots n$ **do**
11: **if** $\widetilde{t}_j \neq 0$ **then**
12: $Y_j \leftarrow Y_j + \widetilde{Y}_j / \widetilde{t}_j$
13: $c_j \leftarrow c_j + \widetilde{c}_j / \widetilde{t}_j$
14: $t_j \leftarrow t_j + 1$
15: **end if**
16: **end for**
17: **end loop**

matrix Y will be shared among all the devices. This is safe, because matrix Y contains only user-independent information about all the items the users might consume.

The gradient of Y computed by a single user may leak private data. In federated learning, Bonawitz et al. [3] describe a secure aggregation protocol with additional measures to prevent this kind of information leakage. In gossip learning, one can apply the secure distributed mini-batch methods described in [5]. We do not include such additional techniques in our present study.

Using the loss function defined in Equation (3), and assuming that every device has a copy of Y, the gradient of both its own row of X and the global matrix Y can be computed by each device locally, w.r.t. the local row of A. Devices can use these gradients to update their own row of X locally. Therefore, all we need to take care of is to somehow aggregate the gradients of Y over the devices and then redistribute new versions of Y. Federated learning and gossip learning offer two, rather different alternative solutions to this problem. (Note that the bias vectors b and c are handled similarly to X and Y, respectively.)

3 Federated Learning

Here, we present the well-known federated learning algorithm [10, 12], adapted to the problem of rank-k matrix decomposition.

In this framework, there is a master node that runs Algorithm 1, and several worker nodes that execute Algorithm 2. The master first initializes the global model (t, Y, c) that contains matrix Y, the bias vector c and an *age vector* t. For each row j, t_j counts how many times Y_j and c_j have been updated. Having a separate counter for each row is necessary because there can be a very different

Algorithm 2 Federated Learning Worker

1: $(x_i, b_i) \leftarrow$ initX()
2:
3: **procedure** onReceiveModel($\widetilde{t}, \widetilde{Y}, \widetilde{c}$)
4: $((t, Y, c), (x_i, b_i)) \leftarrow$update($(\widetilde{t}, \widetilde{Y}, \widetilde{c}), (x_i, b_i), a_i$) ▷ a_i: the local ratings
5: $(t', Y', c') \leftarrow (t - \widetilde{t}, Y - \widetilde{Y}, c - \widetilde{c})$
6: send (compress(t', Y', c')) to master
7: **end procedure**

number of examples for each item, and thus there can be a different number of updates applied to each row (see below).

Similarly, each worker node initializes its private model (x_i, b_i) that contains its own row of X, x_i, and the corresponding bias b_i. After initialization, in every round, the master sends the global model to all the workers. The workers then update the received global model and their own local user model using the local ratings, and they then send the (potentially compressed) model gradient to the master. In this message to the master, the vector t' can contain only ones and zeros, indicating which rows of Y (and elements of c) were updated. At the end of each round, the server updates the global model with the average of the received gradients.

Each row Y_j (and value c_j) will typically have different associated $\widetilde{t}_j$ values depending on how many valid (non-missing) values are there in the matrix column A_j, and also on which clients manage to send a message to the master in the given round. We can think of $\widetilde{t}_j$ as the effective mini-batch size corresponding to updating Y_j and c_j. Thus, by normalizing with $\widetilde{t}_j$, we effectively perform parallel mini-batch updates on Y_j and c_j.

Note that in the federated learning framework it is typically assumed that the workers are synchronized; that is, the master has to wait until all (or most of) the nodes send a gradient in the given round and, most importantly, the workers have to wait as well for the next globally aggregated model from the master to process. Although asynchronous distributed learning is common, federated learning also seeks to handle the non-uniform sampling of training data, which is expected to make asynchronous implementations less stable.

The methods update, compress, initX and initY shall be explained in detail in Section 5. Note that the same methods are used in gossip learning as well.

4 Gossip Learning

In gossip learning, there is no master node. All the participants are equivalent, and form a P2P network [14]. All the nodes run Algorithm 3. The nodes first initialize their own copy of the global model (t, Y, c) as well as the private model (x_i, b_i). Then, in each cycle, they send their (potentially compressed) copy of the global model to a random online neighbor in the P2P network. Upon receiving a

Algorithm 3 Gossip Learning

```
1: (t, Y, c) ← initY()
2: (x_i, b_i) ← initX()
3: loop
4:     wait(Δ_g)
5:     p ← selectPeer()
6:     send (compress(t, Y, c)) to p
7: end loop
8:
9: procedure onReceiveModel(t̃, Ỹ, c̃)
10:    (t, Y, c) ←merge((t, Y, c), (t̃, Ỹ, c̃))
11:    ((t, Y, c), (x_i, b_i)) ←update((t, Y, c), (x_i, b_i), a_i)
12: end procedure
```

Algorithm 4 Various versions of the merge function

```
1: procedure mergeNone((t, Y, c), (t̃, Ỹ, c̃))
2:     return (t̃, Ỹ, c̃)
3: end procedure
4:
5: procedure mergeAverage((t, Y, c), (t̃, Ỹ, c̃))
6:     for j ← 1 ... n do
7:         if t̃_j ≠ 0 then
8:             w ← t̃_j / (t_j + t̃_j)
9:             t_j ← max(t_j, t̃_j)
10:            Y_j ← (1 − w)Y_j + wỸ_j
11:            c_j ← (1 − w)c_j + wc̃_j
12:        end if
13:    end for
14:    return (t, Y, c)
15: end procedure
```

model, the node merges it into its own, then updates both the resulting merged new global model and the local model, using the local ratings.

As mentioned above, methods update, compress, initX and initY shall be explained in detail in Section 5. Note that the same methods are used by federated learning as well.

Method merge, however, is specific to gossip learning, and it is responsible for aggregating the updates computed at the devices. Possible implementations of this method are listed in Algorithm 4. The first option is not to perform any aggregation, in which case different versions of the global model perform random walks in the network independently. The other option is to take the average of the two models row by row, weighted by the corresponding elements of the age vectors so that the more converged copy has a larger effect.

An important effect of this weighted merging technique is that the freshly initialized rows of the model of any newly joined node are ignored. This is because

Algorithm 5 Model initialization

```
 1: procedure INITX()
 2:     for d ← 1...k do
 3:         x_d ← rand() · √((R_max − R_min)/k)          ▷ rand() ∼ U(0, 1)
 4:     end for
 5:     b ← R_min/2
 6:     return (x, b)
 7: end procedure
 8:
 9: procedure INITY()
10:     for j ← 1...n do
11:         t_j ← 0
12:         (Y_j, c_j) ←initX()
13:     end for
14:     return (t, Y, c)
15: end procedure
```

if a row has never been updated, then the age is zero for the given row. The new model will be assigned the maximum of the two merged ages. This is a conservative heuristic that performed better in our preliminary experiments than possible alternatives such as the sum of the two ages. Note that the age of the different rows can differ significantly because the number of known ratings for different items typically has a large variance.

Since gossip learning uses a P2P network, we have to make our assumptions about this network explicit. We assume that there is a membership service in our system. This service provides unique identities to the participants that might include public and private keys for public key cryptography that are tied to the network address of the node. The membership service also offers peer sampling, accessed through method SELECTPEER. That is, all the nodes are assumed to have access to addresses of live nodes from the network. In practice, peer sampling can have a decentralized implementation that can be dynamic [16] or it can be based on a static network with random neighbors [15] that is able to handle NAT devices as well. It can also be implemented as a centralized service. Ideally, the neighbors returned by the peer sampling service should be uniform random samples of the live nodes, but in practice it suffices if the network has good mixing when, for example, the neighbors are sampled from a fixed overlay network graph.

5 Shared Methods

Here, we present those methods that are used by both federated learning and gossip learning. Let us begin with the initialization methods in Algorithm 5. Both X and Y are initialized with uniform random numbers from the range $[0, \sqrt{(R_{\max} - R_{\min})/k}]$, and the initial bias is set to $R_{\min}/2$, where $R_{\max}$ and $R_{\min}$ are the largest and the smallest possible ratings, respectively. This ensures

Algorithm 6 Model update rule

procedure UPDATE($(t, Y, c), (x_i, b_i), a_i$)
  for all j where a_{ij} is defined **do**
    $t_j \leftarrow t_j + 1$
    $err \leftarrow a_{ij} - x_i Y_j^T - b_i - c_j$
    $(Y_j, x_i) \leftarrow ((1 - \eta\lambda)Y_j + \eta \cdot err \cdot x_i, (1 - \eta\lambda)x_i + \eta \cdot err \cdot Y_j)$
    $c_j \leftarrow c_j + \eta \cdot err$
    $b_i \leftarrow b_i + \eta \cdot err$
  end for
  return $((t, Y, c), (x_i, b_i))$
end procedure

that when a prediction $x_i Y_j^T + b_i + c_j$ is made using initial values, the result falls in the range $[R_{\min}, R_{\max}]$.

As for learning, both models use a stochastic gradient descent (SGD) update rule with the fixed learning rate η (see Algorithm 6). The age vector t is incremented in positions corresponding to updated rows of Y (that is, for those items that the user rated). The update rule simply follows from the partial derivatives of (3). Note that this version of the update rule uses a constant learning rate, but other implementations might also use the age vector passed to the update method.

Let us now turn to the compression methods. In this study, we focus on subsampling as a simple compression technique. That is, only s rows of Y are sent along with the corresponding elements of t and c, where s is the compression parameter (see Algorithm 7). Subsampling is performed randomly without replacement from the updated rows (that is, those rows where the corresponding rating is known) and, if there is still room left, from the remaining, non-updated rows. Note that sending non-updated rows in fact makes sense because in such cases the given node might act as a forwarding agent. In other words, such rows might be useful for the recipient nodes.

6 Experiments

Here, we present our simulation experiments with gossip learning and federated learning over the MovieLens database in several scenarios.

6.1 Datasets

The MovieLens data sets [8] were collected by the GroupLens Research Project at the University of Minnesota. The data was collected through the MovieLens website (movielens.org) over various periods of time, depending on the size of the set. The main properties of the MovieLens data sets are shown in Table 1 and Figure 1. Each data set is split into a training matrix and a test matrix in such a way that for each user, there are either 0 or 10 defined values in the

Algorithm 7 Various versions of the compress function

```
1: procedure compressNone(t, Y, c)
2:     return (t, Y, c)
3: end procedure
4:
5: procedure compressSubsampling(t, Y, c)
6:     U ← {1, ..., n}
7:     D ← {j ∈ U | a_ij is defined}
8:     J_d ← random subset of D of size min(s, |D|)
9:     J_u ← random subset of (U \ D) of size (s − |J_d|)
10:    for all j ∈ J_d ∪ J_u do
11:        t'_j ← t_j
12:        Y'_j ← Y_j
13:        c'_j ← c_j
14:    end for
15:    return (t', Y', c')          ▷ we assume a sparse vector representation
16: end procedure
```

Table 1. The main properties of the MovieLens data sets and algorithm parameters

	100K	1M	10M
# users (m)	943	6,040	69,878
# movies (n)	1,682	3,952	10,677
# ratings	100,000	1,000,209	10,000,054
Density	6.3%	4.2%	1.3%
Training / Test	90.57% / 9.43%	93.96% / 6.04%	93.01% / 6.99%
Time period	20.09.97 - 22.04.98	25.04.00 - 28.02.03	09.01.95 - 05.01.09
$\eta/\lambda/k$	$10^{-2}/10^{-1}/5$	$10^{-2}/10^{-1}/5$	$10^{-2}/10^{-1}/5$
message size	0.6 Mbit	1.5 Mbit	4.1 Mbit

test matrix. Each row of the training matrix (representing the ratings of a given user) was assigned to a unique node in the simulation experiments.

6.2 System Model

In our simulations, fixed random 20-out graphs were used as the overlay network. The number of nodes was equal to the number of users in the given data set. In the churn-free scenario, every node stayed online for the whole experiment. A real availability trace, gathered from smartphones, was used in the churn scenario. A message was considered successfully delivered if and only if both the sender and the receiver remained online during the transfer. Peer selection (method selectPeer) returned online nodes only.

The nodes had the same upload and download bandwidths. The motivation for this was that it is likely that in a real application there will be a low, uniform,

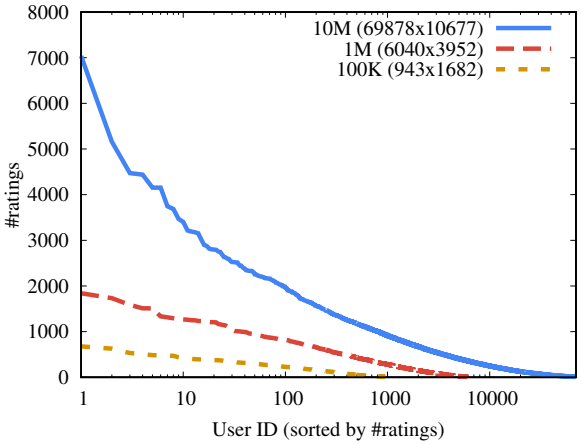

Fig. 1. Visualization of the distribution of the number of rated items per user. Users are sorted according to the number of their rated items for all three databases.

configured bandwidth cap. The server had infinite bandwidth (which favors federated learning, as gossip learning does not use a server). The transfer time of a full model was assumed to be 1728 s (irrespective of the data set used) in the low bandwidth scenario, and 172.8 s in the high bandwidth scenario. This allowed for around 100 and 1000 iterations over the course of 48 hours, respectively.

The cycle length parameters Δ_g and Δ_f were set so that the two approaches fully utilized the available bandwidth. In our case this also means that the two algorithms transfer the same amount of data overall in the network in the same amount of time, making comparisons of convergence dynamics fair. The gossip cycle length Δ_g is exactly the transfer time of a model, which is proportionally smaller when compression is used. The cycle length Δ_f of federated learning is the round-trip time, that is, the sum of the upload and download times. In this case, only the upstream transfer is compressed.

Note that we use rather low bandwidth settings because in the churn scenario if the transfer is very fast, the network hardly changes during the learning process, the models are learned over an effectively static subset of the nodes. Slower transfer is more challenging, because more transfers fail, just like in the case of very large machine learning models such as deep neural networks. (This issue is completely irrelevant in the churn-free scenario, since the dynamics are identical apart from the scale of time.)

6.3 Smartphone Traces

We used a trace collected by STUNner, a locally developed, openly available smartphone application [2]. In short, the app monitors and collects information about the battery level, charging status, bandwidth, and NAT type.

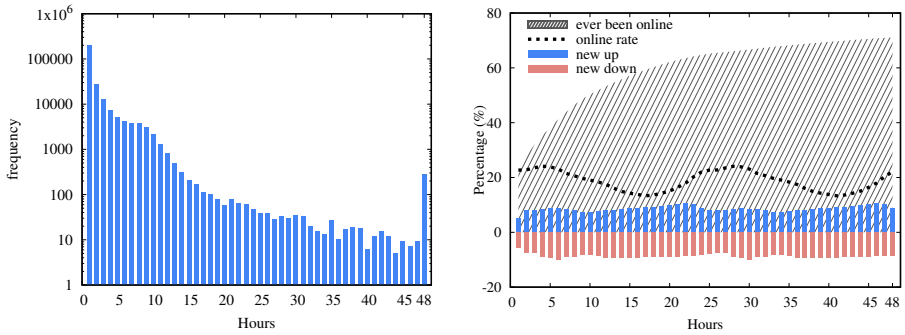

Fig. 2. Online session length histogram (left) and device churn (right).

The trace contains time series spanning varying lengths of time, originating from 1191 different users. Based on the UTC hour of day, we split the data into 2-day segments (with a one-day overlap), resulting in 40,658 segments altogether. Using this, we can simulate a virtual 48-hour period by assigning a segment to each simulated node.

To make our algorithm phone and user friendly, we consider a device to be online (available) when it has been on a charger and connected to the Internet (with a bandwidth of at least 1 Mbit/s) for at least a minute, therefore we do not use battery power at all.

The main properties of the trace are shown in Figure 2. The plot on the right illustrates churn by showing what percentage of the nodes left, or joined the network (at least once) in any given hour. Notice that at any given moment about 20% of the nodes are online. The mean online session length is 81.368 min.

6.4 Hyperparameters

The learning rate η and regularization parameter λ were optimized in the churn-free, low-bandwidth, uncompressed scenario. The resulting values are $\eta = 10^{-2}$ and $\lambda = 10^{-1}$, as shown in Table 1. We used rank-5 factorization.

6.5 Results

We used PeerSim [13] for the simulations. We measured performance with the help of the root-mean-square deviation

$$\mathrm{RMSE} = \sqrt{\frac{1}{|T|}\sum_{i,j\in T}(r_{i,j} - x_i y_j^T)^2},$$

where $R \in \mathbb{R}^{m\times n}$ is the test matrix, and T is the set of indices defined in R. In the case of gossip learning, the error is calculated using the models stored in the currently online nodes and the corresponding rows of R. In the case of federated

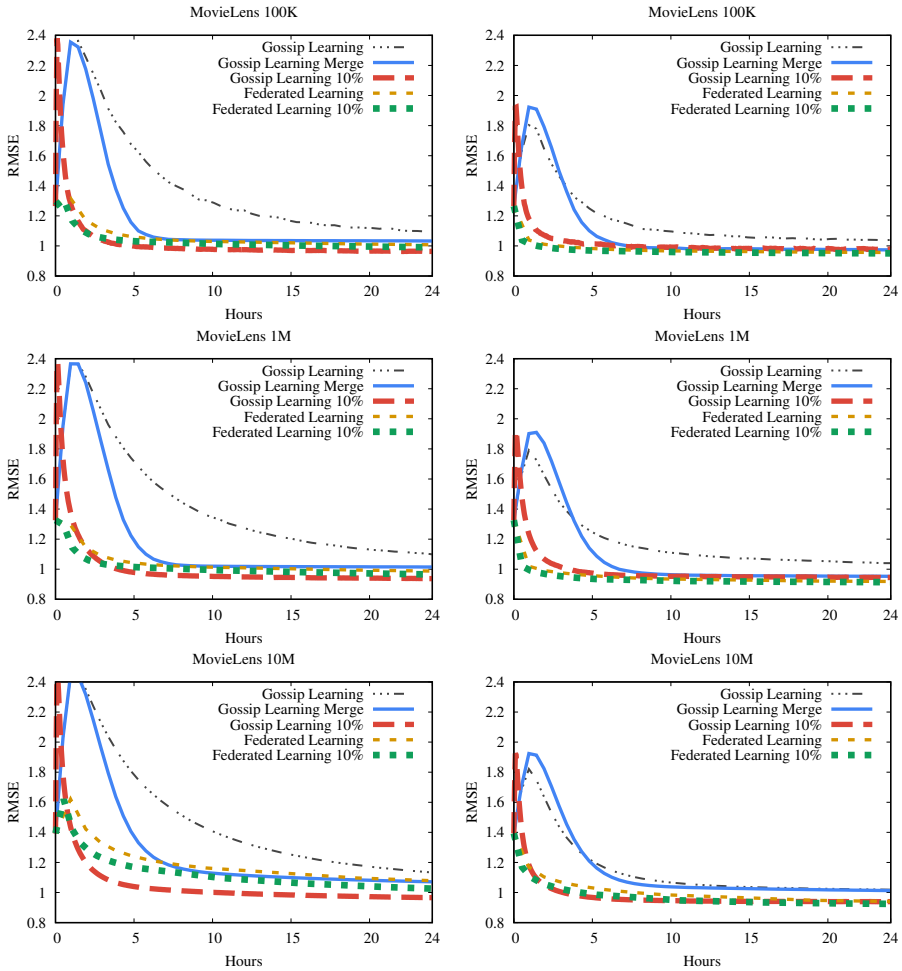

Fig. 3. Churn-free scenario with 1 epoch (left) and 10 epochs (right).

learning, the aggregated global model is used instead of the local ones. Figure 3 contains our results without churn, and Figure 4 shows the same experiments over the smartphone trace. The evaluated algorithms are

Gossip Learning: no merging and no subsampling. Here, the cycle length equals the time needed for one full model transmission.

Gossip Learning Merge: with merging but no subsampling, so the cycle length is still one full transmission.

Gossip Learning 10%: with merging and subsampling with $s = n/10$. Here, the cycle length corresponds to 0.1 full transmissions.

Federated Learning: no subsampling, so the cycle length corresponds to two full transmissions: upload and download.

Federated Learning 10%: the uploaded model is subsampled with $s = n/10$, so the cycle length corresponds to 1.1 full transmissions.

In Figure 3 we include results with 1 and 10 epochs of local learning in the left and right columns, respectively. In the case of 10 epochs, the local gradient update step is iterated 10 times. Clearly, for both methods, increasing the number of epochs improves convergence speed without any extra communication. The compressed variants consistently perform better. Federated learning has an initial advantage, which disappears after a few hours. In fact, for the largest problem, gossip learning is almost identical to federated learning, and the difference between the two methods seems to decrease with increasing network size. Interestingly, when only 1 epoch is performed, gossip learning actually outperforms federated learning by a significant margin especially on the largest network.

In Figure 4 we ran one local epoch in each experiment, but the plots on the right show the effect of speeding up communication. Faster communication results in a dramatically better performance, simply because the convergence speed is able to "beat" the speed of churn. Apart from this observation, the other conclusions are similar, namely compression helps both methods and gossip learning performs relatively better in larger networks. Overall, federated learning and gossip learning have a very similar performance, despite the disadvantage of gossip learning of not relying on a central server for aggregation and broadcast.

7 Conclusions

In this study, our main goal was to explore the differences between federated learning and gossip learning over a collaborative filtering task. Since gossip learning does not rely on central servers, one might expect it to pay a performance penalty in terms of convergence speed, when given the same communication budget.

Our main conclusion based on our empirical study is that federated learning does not seem to have a clear performance advantage. In fact, in certain scenarios gossip learning proved to be preferable. Obviously, the design space for both protocols is very large, and there are many possibilities for improving the communication efficiency in both paradigms. It is also a non-trivial question of how one should model communication constraints and costs, since this depends on many factors. However, it is interesting, and perhaps non-trivial, that gossip learning is clearly comparable in performance. This might motivate further research into fully decentralized methods that otherwise have clear benefits such as a very low cost of entry that is not dependent of the network size, or the robustness due to the lack of any critical components.

References

1. Ammad-ud-din, M., Ivannikova, E., Khan, S.A., Oyomno, W., Fu, Q., Tan, K.E., Flanagan, A.: Federated collaborative filtering for privacy-

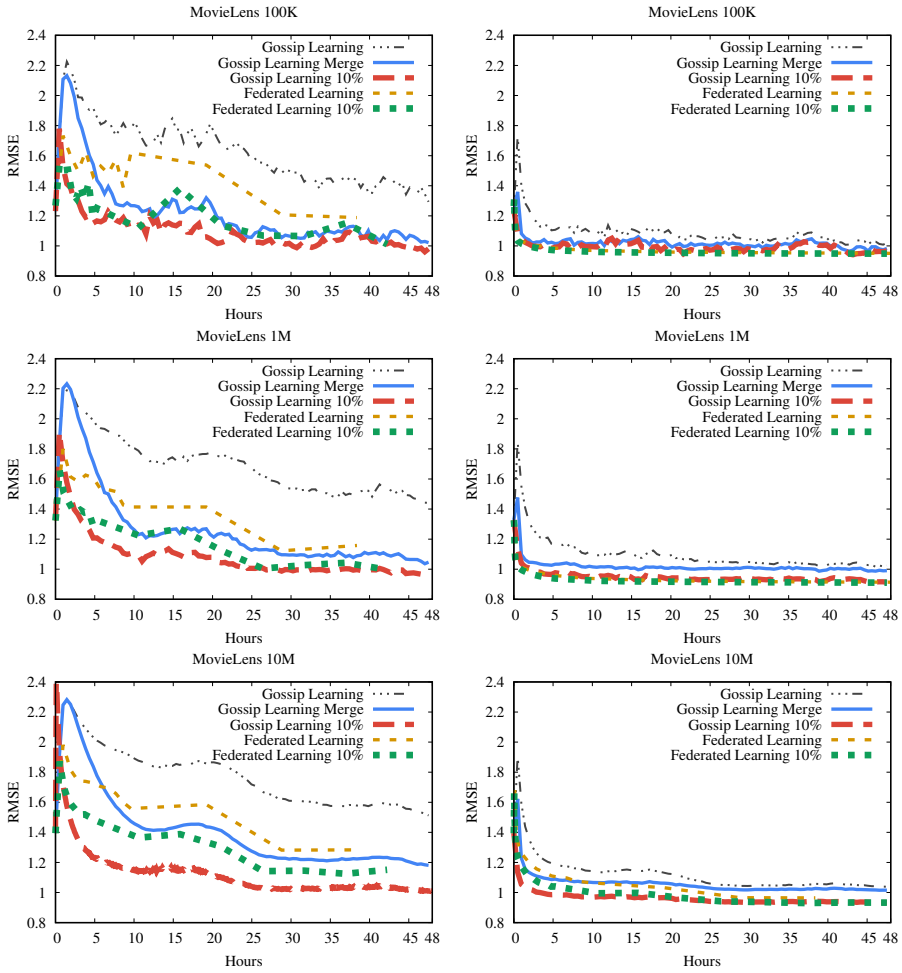

Fig. 4. Churn trace scenario with low bandwidth (left) and 10×higher bandwidth (right).

preserving personalized recommendation system. CoRR **abs/1901.09888** (2019), http://arxiv.org/abs/1901.09888

2. Berta, Á., Bilicki, V., Jelasity, M.: Defining and understanding smartphone churn over the internet: a measurement study. In: Proceedings of the 14th IEEE International Conference on Peer-to-Peer Computing (P2P 2014). IEEE (2014). https://doi.org/10.1109/P2P.2014.6934317
3. Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H.B., Patel, S., Ramage, D., Segal, A., Seth, K.: Practical secure aggregation for federated learning on user-held data. In: NIPS Workshop on Private Multi-Party Machine Learning (2016)
4. Chen, F., Dong, Z., Li, Z., He, X.: Federated meta-learning for recommendation. CoRR **abs/1802.07876** (2018), http://arxiv.org/abs/1802.07876

5. Danner, G., Berta, Á., Hegedűs, I., Jelasity, M.: Robust fully distributed minibatch gradient descent with privacy preservation. Security and Communication Networks **2018**, 6728020 (2018). https://doi.org/10.1155/2018/6728020
6. Dean, J., Corrado, G.S., Monga, R., Chen, K., Devin, M., Le, Q.V., Mao, M.Z., Ranzato, M., Senior, A., Tucker, P., Yang, K., Ng, A.Y.: Large scale distributed deep networks. In: Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1. pp. 1223–1231. NIPS'12, Curran Associates Inc., USA (2012), http://dl.acm.org/citation.cfm?id=2999134.2999271
7. European Commission: General data protection regulation (GDPR) (2018), https://ec.europa.eu/commission/priorities/justice-and-fundamental-rights/data-protection/2018-reform-eu-data-protection-rules
8. Harper, F.M., Konstan, J.A.: The movielens datasets: History and context. ACM Trans. Interact. Intell. Syst. **5**(4), 19:1–19:19 (Dec 2015). https://doi.org/10.1145/2827872, http://doi.acm.org/10.1145/2827872
9. Hegedűs, I., Berta, Á., Kocsis, L., Benczúr, A.A., Jelasity, M.: Robust decentralized low-rank matrix decomposition. ACM Transactions on Intelligent Systems and Technology **7**(4), 62:1–62:24 (May 2016). https://doi.org/10.1145/2854157
10. Konecný, J., McMahan, H.B., Yu, F.X., Richtárik, P., Suresh, A.T., Bacon, D.: Federated learning: Strategies for improving communication efficiency. In: Private Multi-Party Machine Learning (NIPS 2016 Workshop) (2016)
11. Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer **42**(8), 30–37 (2009). https://doi.org/10.1109/MC.2009.263
12. McMahan, B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data. In: Singh, A., Zhu, J. (eds.) Proceedings of the 20th International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 54, pp. 1273–1282. PMLR, Fort Lauderdale, FL, USA (20–22 Apr 2017)
13. Montresor, A., Jelasity, M.: Peersim: A scalable P2P simulator. In: Proceedings of the 9th IEEE International Conference on Peer-to-Peer Computing (P2P 2009). pp. 99–100. IEEE, Seattle, Washington, USA (Sep 2009). https://doi.org/10.1109/P2P.2009.5284506, cikkek/p2p09.pdf, extended abstract
14. Ormándi, R., Hegedűs, I., Jelasity, M.: Gossip learning with linear models on fully distributed data. Concurrency and Computation: Practice and Experience **25**(4), 556–571 (2013). https://doi.org/10.1002/cpe.2858
15. Roverso, R., Dowling, J., Jelasity, M.: Through the wormhole: Low cost, fresh peer sampling for the internet. In: Proceedings of the 13th IEEE International Conference on Peer-to-Peer Computing (P2P 2013). IEEE (2013). https://doi.org/10.1109/P2P.2013.6688707
16. Tölgyesi, N., Jelasity, M.: Adaptive peer sampling with newscast. In: Sips, H., Epema, D., Lin, H.X. (eds.) Euro-Par 2009. Lecture Notes in Computer Science, vol. 5704, pp. 523–534. Springer-Verlag (2009). https://doi.org/10.1007/978-3-642-03869-3_50
17. Wang, J., Cao, B., Yu, P.S., Sun, L., Bao, W., Zhu, X.: Deep learning towards mobile applications. In: IEEE 38th International Conference on Distributed Computing Systems (ICDCS). pp. 1385–1393 (Jul 2018). https://doi.org/10.1109/ICDCS.2018.00139, https://arxiv.org/pdf/1809.03559.pdf